第4章

ASICをターゲットに，ビヘイビア合成で作成したRTLコードを論理合成

── 配置配線担当者との円滑なコミュニケーションが不可欠

横溝憲治

ここではビヘイビア合成ツールを使って作成したRTLコードを論理合成し，ASIC化するまでの手順を説明する．サンプルとして，前章で作成したJPEGエンコーダの設計データを利用する．ASICを開発する場合，テスト容易化設計やASICサインオフのしくみなどを理解する必要がある．ASICベンダの配置配線担当者などとコミュニケーションをとりながら，作業を進めることが肝要である．　（編集部）

ここでは，ビヘイビア合成ツールによって生成されたRTL（register transfer level）記述をLSI化する作業手順とその事例について説明していきます．サンプルとしては，前章で作成したRGB-YUV変換とDCT（離散コサイン変換）のモジュールのRTL記述を使用します．

カスタムLSIを実現する方式には，大きく分けてASIC（application specific integrated circuit）とFPGA（field programmable gate array）があります．ビヘイビア合成技術はおもに回路規模や量産数量の大きいASICの開発に利用されているため，本稿ではASIC化を想定しながら，事例などを説明します．FPGAしか開発したことがない機器設計者にも理解しやすいように，ASIC設計とFPGA設計の違いを対比しながら解説します．

表1　ASICとFPGAの比較

	ASIC	FPGA
回路のプログラミング	不可	可能
回路規模	数千万ゲート	数百万ゲート
ゲート当たりの面積	小さい	大きい
動作速度	速い	遅い
消費電力	小さい	大きい
設計・検証期間	長い	短い．早期に実デバイスで検証可能
製造期間	数週間	なし
開発費	高い	安い
チップ単価	安い	高い

● ASIC vs. FPGA

ASICとFPGAの特徴を表1に示します．最大の違いは，FPGAはプログラミング（書き換え）可能なことです．最大回路規模，動作速度，消費電力など，性能面に着目するとASICのほうが有利です（回路をプログラミングできるようにしているため，FPGAにはいくつかの制限がある）．

以前は製品のプロトタイプなどに限ってFPGAが採用されましたが，近年はFPGAの大規模化や低価格化，ASICの開発費の高騰により，トータル・コスト（開発費とデバイス購入費）を考えてFPGAを採用するケースが増えています．しかし，FPGAでは求める性能が得られない場合には，ASICを選択することになります．

FPGAを使用した場合，機能の差異化はできますが，性能の差異化はそれほどできません．性能面で特徴を出したい場合には，FPGAよりASICのほうが有利です．

● トータル・コストは開発費とチップの購入数で決まる

ASICとFPGAのトータル・コストとチップの購入数の関係は図1のようになります．

回路規模が同じである場合，チップ単価はASICのほうが安く，FPGAのほうが高くなります．一方，開発費については，FPGAではベンダに支払う開発費が存在しないため，ユーザ側の設計費用のみとなります．ASICでは，ユーザ側の設計費用に加えて，チップごとに必要になるマスクの作成費用など，ASICベンダへ支払う開発費が必要になります．

トータル・コストは，チップの購入数が少ないときは

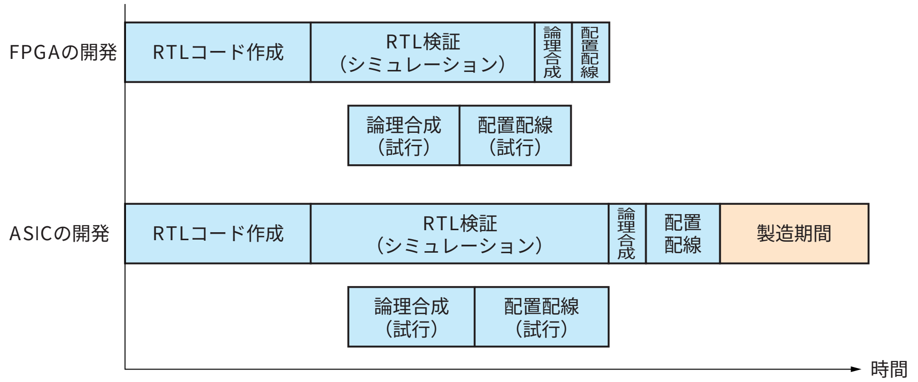


図2
ASICとFPGAの開発期間
FPGAは早期にデバイスの検証が可能なため，検証期間が短くなる．

FPGAのほうが低くなり，購入数が多くなるとASICのほうが低くなります．つまり，ASICを利用するか，FPGAを利用するかは購入数によって選択することになります．

分岐点となる購入数は，回路規模によって変わります．FPGAの価格は回路規模が大きいほどより割高になるため，回路規模が大きくなるとASICを選択する傾向があります．また，ASICの開発費は半導体プロセスの微細化にともなって高額になり，その一方でFPGAのゲート当たりの単価は下がり続けています．そのため，年を追うごとに，分岐点となる購入数は図1の右側へシフトしているようです．

実際のところ，製品のライフ・サイクルが短くなってきていることを考えると，開発費の高いASICを選択できないケースが増えています．

● ASICの開発期間は長びく傾向に

ASICとFPGAの開発期間の概要を図2に示します．ここでは開発期間を設計・検証期間と製造期間の二つに分けて考えます．

設計・検証期間については，本来であればASICとFPGAにほとんど差はないはずです．しかし，設計修正が発生した場合に，FPGAならすぐに実際のデバイスを動作させて検証できるため，手間のかかるシミュレーションが省略される傾向にあります．一方，ASICは一度製造すると作り直しに多くの費用と製造期間が必要になるので，より慎重に論理検証やタイミング検証が実施され，設計・検証期間が長びく傾向にあります．

製造期間については，FPGAは設計完了後，数分～数時間で回路を実現できます．一方，ASICは設計完了後，数週間が必要です．製造後，設計上の不ぐあいが見つかった場合には，製造期間の違いはさらに大きくなります．

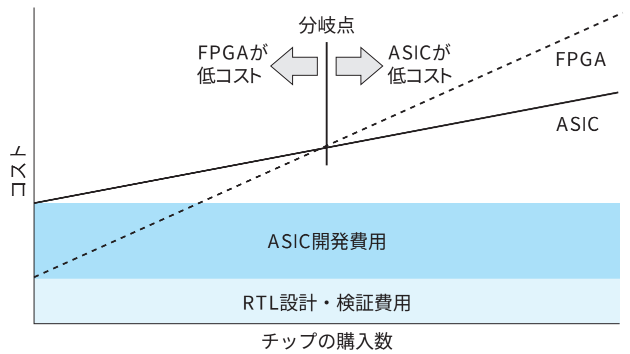


図1　トータル・コストとチップの購入数
購入数が少ないと，マスクなどの開発費が不要なFPGAのほうがコストが低くなる．購入数が多くなると，チップ単価が安いASICのほうがコストが低くなる．

● FPGAとのギャップを埋めるストラクチャードASIC

ASICが抱える開発費と製造期間の課題を緩和するため，開発費を安価に抑え，製造期間を短縮したストラクチャードASICというLSIが登場しています．おもしろいことに，ストラクチャードASICの製品はASICベンダとFPGAベンダの双方から出荷されています．双方とも，ASICとFPGAの間のギャップを埋めることが新たなビジネス・チャンスととらえているようです．

FPGAベンダのストラクチャードASICは，FPGA用の設計データをそのままストラクチャードASICの開発に流用できることを特徴としています．使用するツールもFPGA開発ツールと共通になっています．チップが大量に必要になった場合，FPGAからストラクチャードASICへスムーズに移行できます．

一方，ASICベンダは，レイアウト設計（配置配線設計）で手間のかかる部分の配線をあらかじめ済ませることにより，設計期間の短縮と個別に作成するマスクの枚数を減ら

4

したASICをストラクチャードASICと位置づけています．ASICベンダによって構造はまちまちです．ゲートアレイはチップ全面に敷き詰めたゲートの間を配線しますが，ストラクチャードASICではRAMブロック，各種IPコア，クロック・ツリーなどを下地として用意し，その間を配線します．そのため，レイアウト設計の負担が軽くなります．また，下地として用意したDフリップフロップ（DFF）や基本論理セルを配線するタイプの製品もあります．

● ASIC開発フローには二つの関門がある

ASICの設計フローを見ていきましょう．**図3**はASICの設計フローの例です．

RTL記述の作成が終わった段階で，シミュレーションによるRTL記述の機能検証と並行して，論理合成の作業を進めます．最初にフロアプラン（ピン配置や回路ブロックの概略配置）を検討し，タイミング制約を設定して論理合成ツールでゲート・レベル・ネットリストを生成します．論理合成の結果として得られた予測配線遅延をもとにタイミング・チェックや遅延シミュレーションを行い，さらに等価性検証（RTL記述とゲート・レベル・ネットリストの間に論理的な不整合がないかのチェック）を行います．これらの結果に問題がなければ，ファースト・サインオフ（ASICユーザからLSI製造会社へ設計データを引き渡すときの第1関門）が通過となり，ASICベンダに配置配線を実施してもらいます．

配置配線が終わったら，その結果（実配線長を考慮した配線遅延など）を反映して再度タイミング・チェックや遅延シミュレーション，等価性検証を行います．これらに問

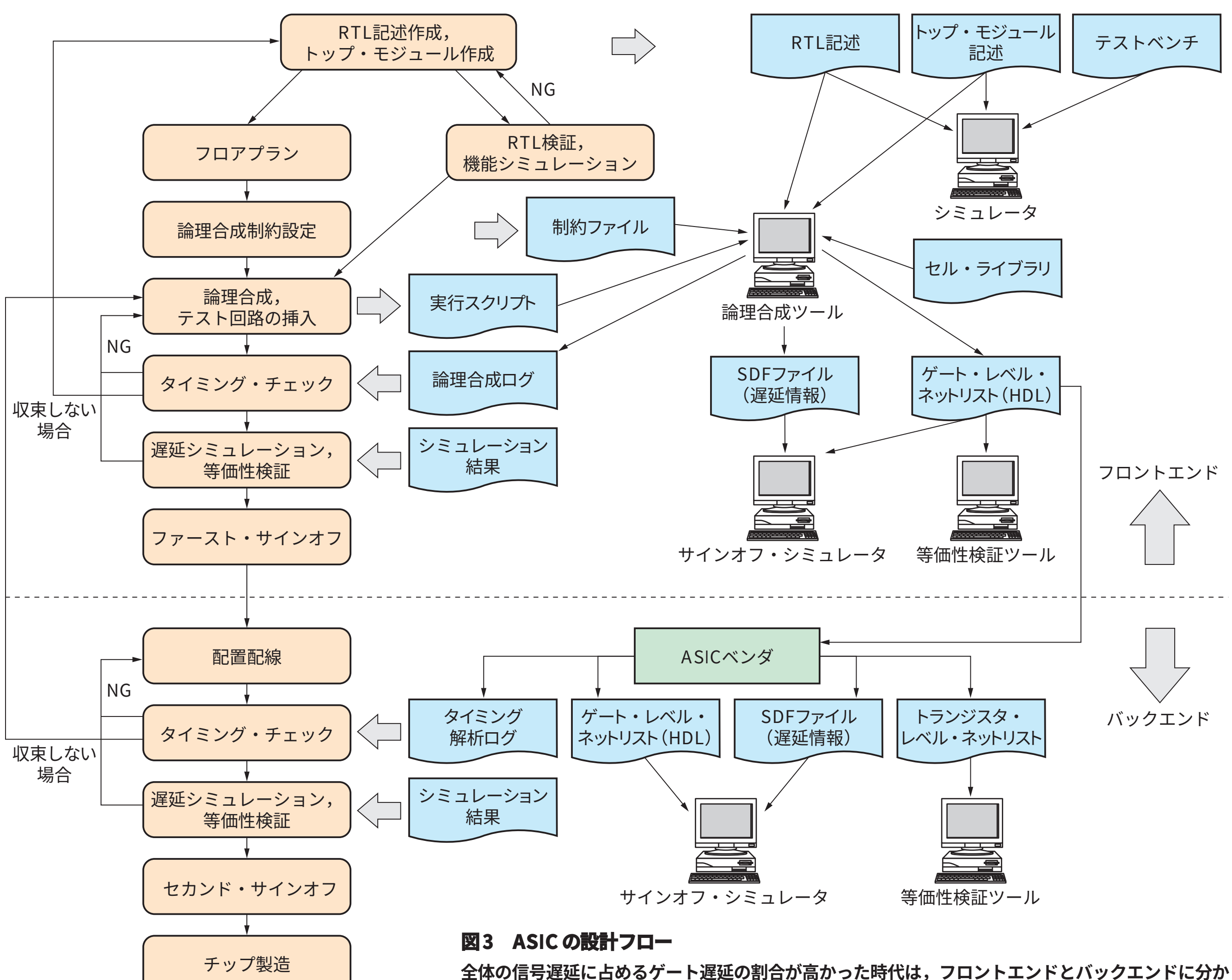


図3　ASICの設計フロー

全体の信号遅延に占めるゲート遅延の割合が高かった時代は，フロントエンドとバックエンドに分かれた作業分担がうまくいっていた．近年は，プロセスの微細化にともない，配線遅延の割合が高くなったため，フロントエンドとバックエンドを完全に分けて作業を進めることが難しくなっている．

題がなければセカンド・サインオフ(製造直前の最終関門)通過となって，ASICを製造してもらえます．

チップは初めに少数のES(engineering sample)が提供され，ESを使って動作確認を行った後，量産に移ります．

一般に，論理合成完了までの作業をフロントエンド，配置配線以降の作業をバックエンドと呼ぶことが多いようです．FPGAはすべての工程の作業をユーザ側で行うのに対して，ASICではフロントエンドはユーザが担当し，バックエンドは配置配線などの専門的な知識が必要になるのでASICベンダが担当します．また，RTL記述をASICベンダに渡して論理合成の工程も委託する「RTLサインオフ」というサービスもあります．当然，その分の費用はASICベンダに追加で支払うことになります．

以下ではそれぞれの工程の作業について説明していきます．

● ピン配置と回路ブロックの配置を検討

フロアプラン工程では，ASICのピン配置と内部の回路ブロックの配置を検討します．

まず，ASICを実装するボード(プリント基板)のレイアウトとASICベンダが取り決めたピン配置のガイドラインを参考にして，ピン配置を決定します．次に，ピン配置を参考にしながら内部のブロックの配置を検討します．一般には配線の接続が多いブロックどうしを隣接するように配置します．配置にはデバイスごとにさまざまなノウハウがあるので，ASICベンダの配置配線担当者とよく相談して決めることが肝要です．

● トップ・モジュールを作成

ASICではRTL記述の階層の上にトップ・モジュールを作成します．トップ・モジュールにはRTL記述モジュール，I/Oバッファ，IPコアなどのインスタンスを記述します．デバイスに依存するI/Oバッファなどをトップ・モジュールに記述し，RTL記述にはデバイスに依存する記述をなくします．これにより，RTL記述がいろいろな設計で流用しやすくなります．

本稿でサンプルとする二つのモジュール(RGB-YUV変換，DCT)もサブモジュールとしてインスタンスします．図4はトップ・モジュールのようす，リスト1はトップ・モジュール記述のI/Oバッファと RTL記述のインスタンス部分です．

テスト回路の挿入とスキャン・パス用テスト端子への接続はツールで行うので，テスト端子は空き端子になっています．I/Oバッファは，論理合成ツールに挿入させることも可能です．ツールで挿入する場合は，I/Oバッファのインスタンスを記述しません．

また，FPGAでプロトタイプを作成する場合，FPGA用のトップ・モジュールを別に作成します．FPGAでは，多くの場合，I/Oバッファをインスタンスしません．テスト端子の取り扱いも異なります．RTL記述はASICと共通のソースを使用します．

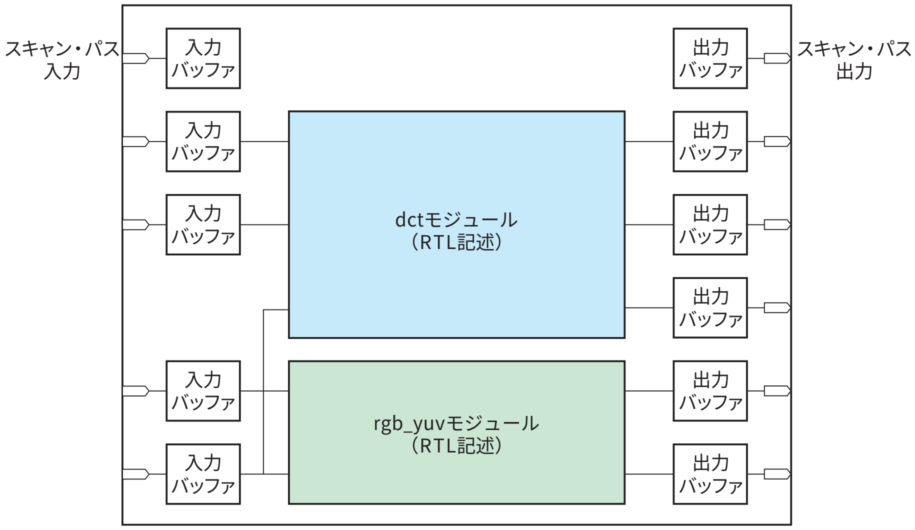


図4　トップ・モジュール

スキャン回路の挿入と接続は論理合成時に行うので，スキャン・パス用端子は接続先のない空き端子になっている．RTL記述は下位階層としてインスタンスされている．

リスト1　トップ・モジュール記述の一部

```
// 入力バッファのインスタンス
  PDUWDGZ scan_in_inbuf (
      .C(scan_in_pad),
      .PAD(scan_in),
      .REN(gnd_sig));

// 出力バッファのインスタンス
  PDO02CDG scan_out_obuf (
      .PAD(scan_out),
      .I(scan_out_pad));

// dct モジュールのインスタンス
dct dct(
.clock(         clock_pad),
.reset(         reset),
.start(         dct_start_pad),
.done(          dct_done_pad),
.uIB1_address( uIB1_address),
.uIB1_q(        uIB1_q),
.uIB1_data(     uIB1_data),
.uIB1_we(       uIB1_we),
.uIB2_address( uIB2_address),
.uIB2_q(        uIB2_q),
.uIB2_data(     uIB2_data),
.uIB2_we(       uIB2_we)
 ) ;
```

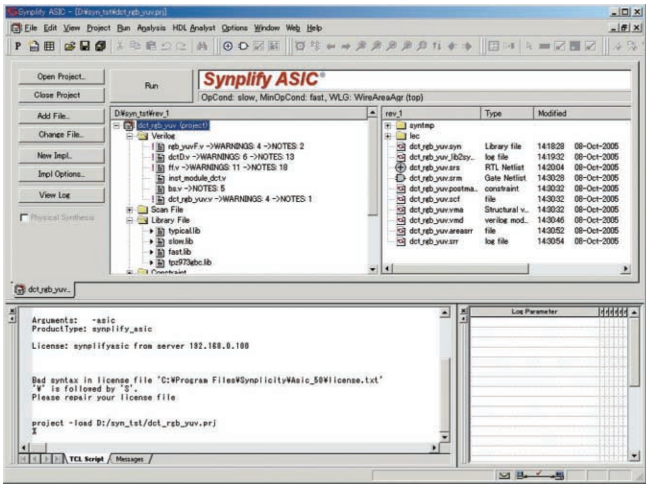


図5　論理合成ツール（Synplify ASIC）の画面

上段の左側のウィンドウには，ツールに読み込むファイルが表示されている．LibraryFileには，ASICベンダのセル・ライブラリを登録する．上段の右側のウィンドウには，ツールが出力したファイルが表示されている．下段のウィンドウでは，実行中のメッセージ表示とコマンド・ラインの操作を行える．

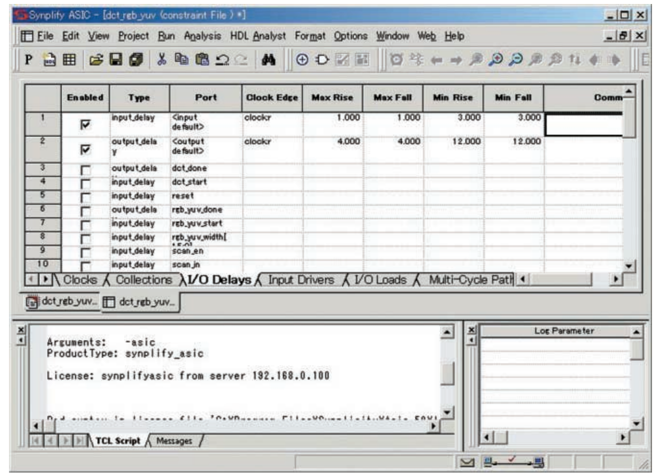


図6　論理合成制約の設定

入出力信号のタイミング制約の設定のようすを示した．設定はSDCファイルに保存される．直接，SDCファイルをテキスト編集して，設定を変更することもできる．

● 制約を与えずに予備的な論理合成を実施

今回は，論理合成ツールとして米国Synplicity社の「Synplify ASIC」を使いました（**図5**，Synplify ASICはhttp://www.synplicity.co.jp/downloads/から評価版をダウンロードできる）．実際の設計で使用する論理合成ツールは何でもよいわけではなく，ASICベンダが指定しているツールを使用する必要があります．

論理合成ツールに入力するデータは，チップのソース・コード，ASICのセル・ライブラリ，制約の設定ファイルです．チップのソース・コードはRTL記述とトップ・モジュール記述です．セル・ライブラリはASICベンダから提供されます．ここでは，英国ARM社が同社のWebサイト上で公開しているArtisan評価ライブラリの台湾TSMC（Taiwan Semiconductor Manufacturing Co.）用スタンダード・セル・ライブラリを使用しました（http://www.artisan.com/products/standard_cell.htmlを開き，左上の「Browse Products:」の「By Foundry」から「TSMC」を選択するとダウンロードできる）．

まずは制約を与えずに論理合成を実行してみます．RTLコードの中に論理合成に適さない記述があった場合，エラーやワーニングが出ます．その場合はRTL記述を修正する必要があります．今回はビヘイビア合成ツールが出力したRTLコードを使いましたが，エラーは出ませんでした．

● 合成制約の設定ファイルを作成

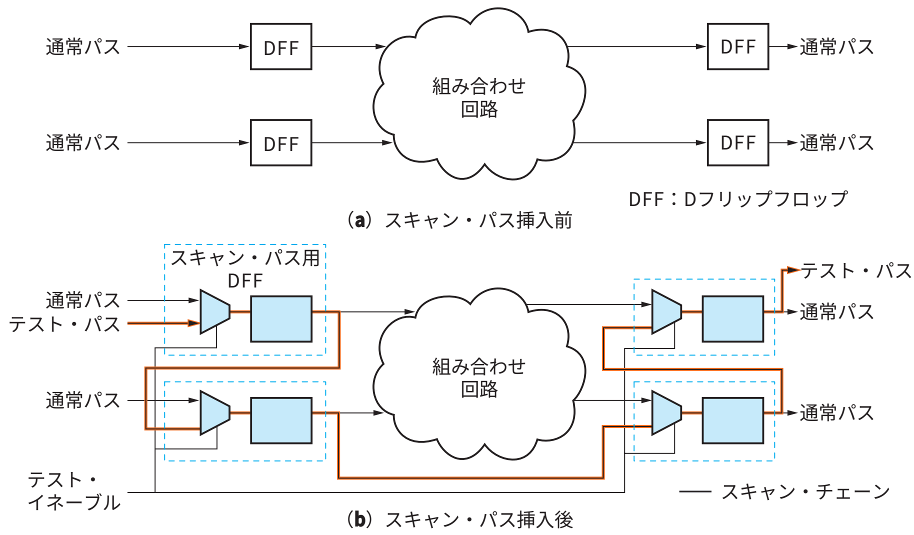
本格的な論理合成を始める前に，ASICに求められる性能に基づいて，合成時の制約を決定します．論理合成ツールは，この制約を満足するように論理回路を生成します．

制約の設定項目は，動作周波数，入出力タイミング，面積などになります．また，タイミング制約を無視してもよいパス（例えば，実際に動作したときに活性化しないパスやタイミング解析ツールが処理できない特殊なパスなど）についてはフォルス・パス設定を行います．**図6**は今回使用した論理合成ツールの入出力タイミング制約設定の画面です．設定はSDCファイルに保存されます．SDCファイルをテキスト編集して制約を変更することも可能です．

● スキャン回路を挿入

DFT（design for testability）は，ASICの出荷テストを容易にするためのしくみの総称です．出荷テストはASICの機能的な動作確認ではなく，製造したASICに欠陥がないかどうかを調べるテストです．現在のように大規模化したASICでは，機能を確認する信号パターンを使ったテスト（ファンクション・テスト）だけではASIC全体を検査できません．そこで，ASICの内部にテストのための回路を埋め込み，出荷テストを行っています．

DFTにはスキャン・パス・テスト，BIST（built-in self-test），I_{DDq}テストなどがあります．EDAツールなどでDFTと呼んだ場合，スキャン・パス・テストのみを指して



◀ **図7**
スキャン回路
スキャン・テストではDFFが，マルチプレクサを備えるスキャン・パス用DFFに置き換えられる．テスト・パスは自動で接続される．

リスト2　スキャン・パス用テスト・ピンの設定

```
define_test_chain 1 ¥
  -test_enable scan_en ¥
  -test_in scan_in ¥
  -test_out scan_out
```

いることもあります．ここではスキャン・パス・テストについて詳しく説明します．

スキャン・パス・テストは，ASIC設計ではかならず採用されます．スキャン・パス・テストのしくみは図7のようになります．DFFの手前にマルチプレクサを挿入して，通常のパスとテスト・パスを切り替えられるようにします．テスト・パスは，DFFがちょうどシフト・レジスタになるように構成されています．

テストでは，初めにテスト・モードでDFFをシフト・レジスタにして，外部から値を設定します．次に，通常モードに切り替えて，組み合わせ回路の出力値をDFFに取り込みます．そして，再びテスト・モードに戻してDFFをシフト・レジスタにして，値を外部に取り出します．

スキャン回路は，論理合成ツールなどを使って挿入します．ただし，スキャン・パス・テストに対応するため，対象となるRTL記述にはクロックの取り扱いなどに制約があります．スキャン回路を挿入するツールには制約をチェックする機能があるので，RTL記述がコンパイル可能となった時点でチェックを行ってください．

今回使用する論理合成ツールにもスキャン回路の挿入機能が用意されていたので使ってみました．設定ファイルはリスト2のようなSCNファイルです．このファイルにスキャン・パス用の端子を指定します．論理合成のオプションでUse Scan Cells for TestとStitch Scan Chainsを設定します．論理合成実行時にスキャン・パス用DFFがマッピングされてスキャン・チェーンが構成されます．

図8はスキャン・パスが挿入された回路のようすです．

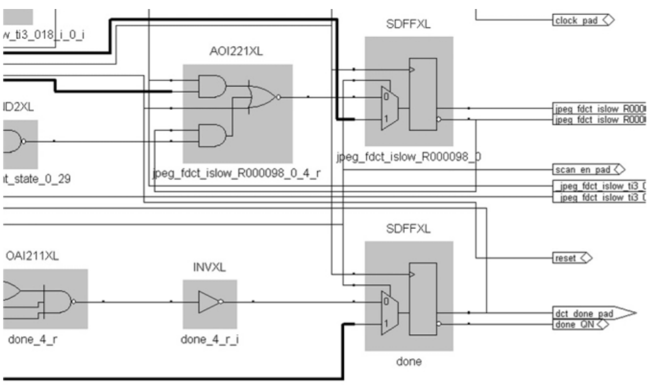


図8　スキャン・パス挿入後の回路
スキャン・パス挿入後のネットリストを論理合成ツールのビューワ(Technology View)で表示した．DFFはスキャン・パス用セル(SDFFXL)に置き換わっている．

スキャン・パス用のテスト・パターンはATPG(automatic test pattern generation)ツールによって生成します．

● 論理合成を実施

論理合成を実行してみましょう．各種設定が終わっていれば，画面の[RUN]ボタンをクリックすることで論理合成が実行されます．論理合成では，オプション設定を変更しながら繰り返し試行し，最適な回路を生成します．スクリプトを使用すると，設定と実行を自動化できます．同じ操作を繰り返さなくて済むので，論理合成の実行にはスクリプトが多用されます．リスト3はスクリプトの例です．

論理合成の結果はログに残されるので，この内容を確認します．図9は今回の実行ログの一部です．動作周波数の見積もりは51.3MHzで，制約の周波数より若干高くなって

います．これはクロック周波数の制約を満たしながら，可能な限り合成後の面積を小さくした結果です．論理合成可能な最高動作周波数を知りたい場合は，制約として高めのクロック周波数を設定して，制約を満たせない状態で論理合成の結果を確認します．合成結果で見積もられた動作周波数が予定しているクロック周波数を大幅に下回っていた場合は，論理合成や配置配線で対処するのは困難なため，RTL記述そのものを見直す必要があります．

使用するツールにもよりますが，多くの場合，論理合成時の信号遅延は予測配線に基づいて算出されています．近年，半導体プロセスの微細化によりゲート遅延が小さくなり，信号遅延全体に占める配線遅延の割合が大きくなりました．そのため，予測配線の遅延は精度が低くなっています．配置配線が完了しないと正確な信号遅延（実配線長に基づく信号遅延）がわからないので，最終的な判断は配置配線後の遅延情報を用いて行います．

リスト3　論理合成の実行スクリプト

```
#プロジェクトのロード
project -load D:/syn_tst/dct_rgb_yuv.prj
#スキャン・パス用端子の設定
add_file -scn ./dct_rgb_yuv.scn
#インプリメンテーションの作成
impl -add rev_5 rev_4
#デフォルトの周波数設定
set_option -frequency 50.000000
#制約ファイルの指定
constraint_file -clear
constraint_file -enable D:/syn_tst/dct_rgb_yuv_1.sdc
#論理合成時のオプション指定
set_option -use_scan_for_test 1
set_option -stitch_scan_chains 1
#論理合成の実行
project -run synthesis
```

● 遅延シミュレーションを実施

遅延シミュレーションに必要なデータは，論理合成ツールが出力します．これには，ゲート・レベルのVerilog HDLネットリストまたはVHDLネットリスト，SDFファイル（遅延情報）です．また，ASICベンダが用意したシミュレーション用ライブラリも必要です．

図10は遅延シミュレーションの実行環境です．シミュレータとして，ASICベンダが指定したサインオフ・シミュレータを使用します．テストベンチはRTL記述のものを流用し，RTLシミュレーションと同じ結果が得られれば問題ありません．**図11**はシミュレーション波形です．

シミュレーション以外の論理検証として，等価性検証があります．フォーマル・ベリフィケーション（形式的検証）ツールと呼ばれる専用ツールを利用して，RTL記述とゲート・レベル・ネットリストの論理的等価性を確認します．タイミング・チェック，遅延シミュレーション，等価性検証に問題がなければファースト・サインオフとなり，ASICベンダに配置配線を実施してもらいます．

```
Performance Summary
*******************


Max worst slack in design: 0.523

Total negative slack in design: 0.000 (0/28280 endpoints)

                   Requested     Estimated     Requested     Estimated     Slack              Clock
Starting Clock     Frequency     Frequency     Period        Period        (max analysis)     Type
----------------------------------------------------------------------------------------------------
clock              50.0 MHz      51.3 MHz      20.000        19.477        0.523              declared
====================================================================================================

  省略

-------------------------------------------------------------------------------
Report for cell dct_rgb_yuv.verilog
  Cell usage:
     cell           count         area     count*area     % of area
       SEDFFXL       6207       89.813     557468.050        49.49%
       AOI2BB2XL     2412       23.285      56162.938         4.99%
       INVXL         2220        6.653      14769.216         1.31%
       NAND2XL       2190        9.979      21854.448         1.94%

  省略

       XNOR2X2          1       36.590         36.590         0.00%
       CLKBUFX8         1       23.285         23.285         0.00%

TOTAL               31262                 1126363.262
```

図9
論理合成の実行ログの一部
動作周波数の見積もりは51.3MHzで，制約の周波数より若干高くなっている．

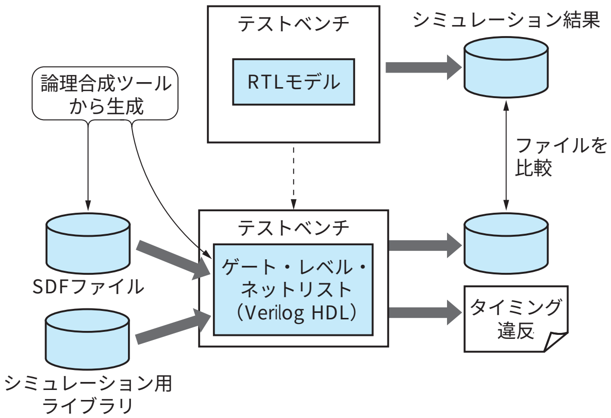


図10　遅延シミュレーションの実行環境

テストベンチについては，RTLシミュレーションで使ったものにSDFファイルの読み込みを追加して流用している．シミュレーション実行後にタイミング違反のレポートを確認する．

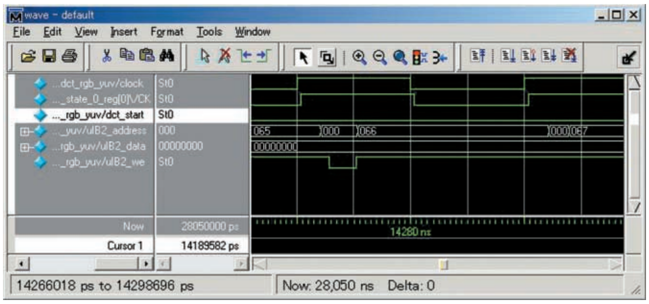


図11　遅延シミュレーションの波形

いちばん上のclockはトップ・モジュールの入力クロックの波形．2番目はDFFのクロック入力の波形．バッファを通過しているので，clockより遅延している．ulB2_addressはバス信号．信号ごとの遅延のばらつきの影響で，clockの立ち上がり後に値がばたついている．ulB2_weは組み合わせ回路の出力なので，ヒゲが発生している．

● ASICベンダに配置配線を依頼

論理合成ツールが出力したゲート・レベル・ネットリストをもとに，ASICベンダが配置配線を行います．ASICベンダは，配置配線のほかに，クロック・ツリー合成やシグナル・インテグリティ（信号品質）解析などの作業を行います．これらは多くのノウハウが必要な作業で，事前に配置配線の試行が不十分だと，この工程で予想外の時間がかかってしまうことがあります．

配置配線が完了すると，正確なタイミングが算出されます．配置配線の変更でタイミング制約を満たせなかった場合は，論理合成の工程に戻ります．制約の設定を変更して新しいゲート・レベル・ネットリストを生成し，再度配置配線を行うことになります．この作業が何度も繰り返されると，開発期間が延びてしまいます．そこで，最近では論理合成と配置配線の機能をあわせ持つフィジカル合成ツールが利用されるようになってきています．

配置配線後のタイミング・チェック，遅延シミュレーション，等価性検証に問題がないことを確認できると，セカンド・サインオフとなります．LSIパッケージの形状や入出力の電気特性など，ASICに必要なすべての情報の確認と認証を行います．

セカンド・サインオフが完了すると，ASICベンダはチップの製造に取りかかります．サインオフした設計データどおりに製造され，出荷テストが行われたチップが供給されます．このチップに動作不良があった場合，ユーザ側の責任になります．

● 設計をスムーズに進めるためには

現在のASIC設計では，プロセスの微細化にともない，レイアウト設計におけるタイミング調整の作業がなかなか収束しなくなっています．フィジカル合成ツールなどで対処が行われていますが，ツールはあくまでも道具でしかありません．ツールを使うエンジニアがどう考えて対処するかが重要となります．

ポイントは，RTL設計担当者，論理合成担当者，配置配線担当者のコミュニケーションだと筆者は思っています．回路の中のDFFを少し増やすだけで，論理合成や配置配線におけるタイミング制約を驚くほど簡単に満足するケースはいくらでもあります．その逆に，速度を上げるためのDFFの追加が難しいときに，配置配線のくふうによって要求速度を達成できる場合もあります．論理合成や配置配線の試行を早期に始めて，問題をどの工程で解決するのがベストなのかを的確に判断することが，ASIC開発を成功させるカギだと思います．

よこみぞ・けんじ
設計コンサルタント

<筆者プロフィール>
横溝憲治．1989年からNECエンジニアリングにてLSI設計および設計ワークステーションのシステム管理に従事．1996年から（有）ひまわりにて，HDL設計のトレーニング講師，受託設計などを請け負い，現在に至る．私生活では地域ボランティア活動に参加．最近はいろいろな意味でPTA活動にはまっている．エンジニアがPTA活動に参加できる道を模索中．

4